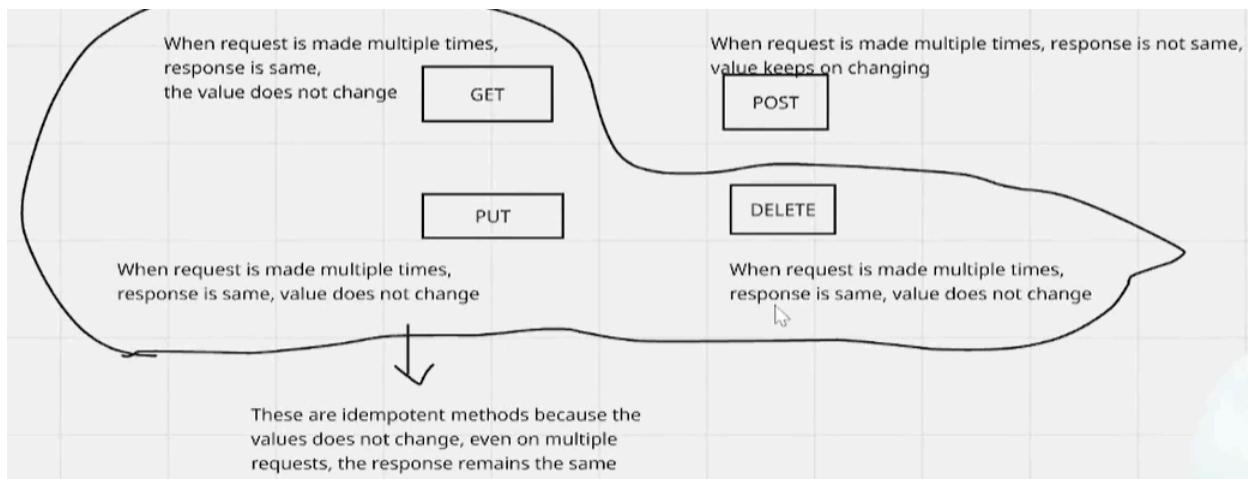


What is idempotency in API Testing?

In the context of APIs and HTTP methods, the term **idempotent** refers to an operation that can be performed multiple times without changing the result beyond the initial application.

In simpler terms:

An idempotent HTTP method is one that has the same effect no matter how many times it is called.



🔍 Examples with HTTP Methods:

HTTP Method	Idempotent?	Explanation
GET	<input checked="" type="checkbox"/> Yes	Fetching the same resource multiple times doesn't change it.
PUT	<input checked="" type="checkbox"/> Yes	Updating a resource with the same data over and over has the same effect.
DELETE	<input checked="" type="checkbox"/> Yes	Deleting the same resource again and again doesn't change the state (once it's gone, it's gone).
POST	<input type="checkbox"/> No	Typically creates a new resource each time it's called — calling it multiple times creates duplicates.

HTTP Methods

GET – "Give me something"

- **What it does:** Asks the server to **send back** data.
 - **Example:** Loading a webpage, viewing a user profile, checking your messages.
 - **Analogy:** Looking at a menu at a restaurant.
-

● **POST** – "Create something new"

- **What it does:** Sends data to the server to **create** something new.
- **Example:** Signing up for an account, making a new post on social media, submitting a contact form.
- **Analogy:** Telling a waiter your order so they can **make it for you**.

● **PUT** – "Replace or update completely"

- **What it does:** Sends data to the server to **replace** something that already exists.
 - **Example:** Updating your full profile (name, email, photo, etc.).
 - **Analogy:** Replacing an old resume with a brand new one.
-

● **PATCH** – "Update a part of something"

- **What it does:** Sends data to the server to **update just part** of something.
 - **Example:** Changing just your profile picture or updating just your email.
 - **Analogy:** Crossing out your old phone number on your resume and writing a new one.
-

● **DELETE** – "Remove something"

- **What it does:** Tells the server to **delete** a resource.
- **Example:** Deleting a tweet, removing an item from your cart, closing your account.
- **Analogy:** Telling the waiter, "Cancel that order."



PUT vs PATCH — Main Difference

Feature	PUT	PATCH
Purpose	Replaces the entire resource	Updates only part of the resource
Idempotent	<input checked="" type="checkbox"/> Yes (safe to repeat)	<input checked="" type="checkbox"/> Usually yes (depends how it's used)
Data needed	Full updated object	Only the fields you want to change
Overwrites?	Yes, it overwrites the entire object	No, it just changes specific fields



Example Scenario:

Let's say you have a user profile like this:

JSON ▾

```
json
CopyEdit
{
  "username": "john_doe",
  "email": "john@example.com",
  "bio": "Loves hiking"
}
```

Using PUT (Full Update)

If you just want to change the `bio`, with `PUT` you'd need to send the **whole object** again, even the parts that didn't change:

```
json
CopyEdit
PUT /user/123

{
  "username": "john_doe",
  "email": "john@example.com",
  "bio": "Coffee addict ☕"
}
```

 Replaces the entire profile with the new version.

Using PATCH (Partial Update)

If you only want to change the `bio`, with `PATCH`, you just send what's changed:

```
json
CopyEdit
PATCH /user/123

{
  "bio": "Coffee addict"
}
```

 Only the `bio` field is updated — the rest stays untouched.

When to Use Which?

- Use `PUT` when you're replacing the **whole resource**.
- Use `PATCH` when you're doing a **partial update**.

What is an Environment in API Testing?

An **environment** is like a **setup** or **workspace** where you test your API.

It includes important things like:

- The **base URL** (where your API is hosted)
- Any **variables** (like API keys, user tokens, etc.)
- Special **settings or data** the API might need

Why do we use environments?

When you're testing an API, it may be running in different places for different purposes:

Environment	What it's for
 Development	Where developers build and test new features
 Testing/QA	Where testers (like you!) test the API safely
 Staging	Like a "dress rehearsal" — final test before going live
 Production	The real thing — where actual users use the API

Real-life Example:

Imagine you're testing a login API:

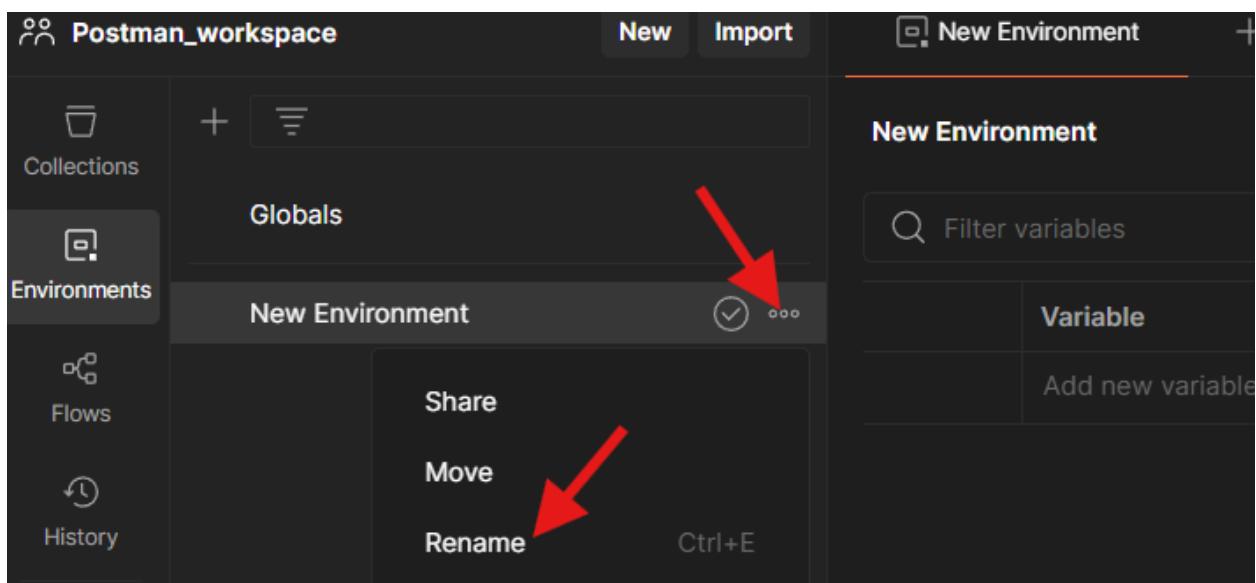
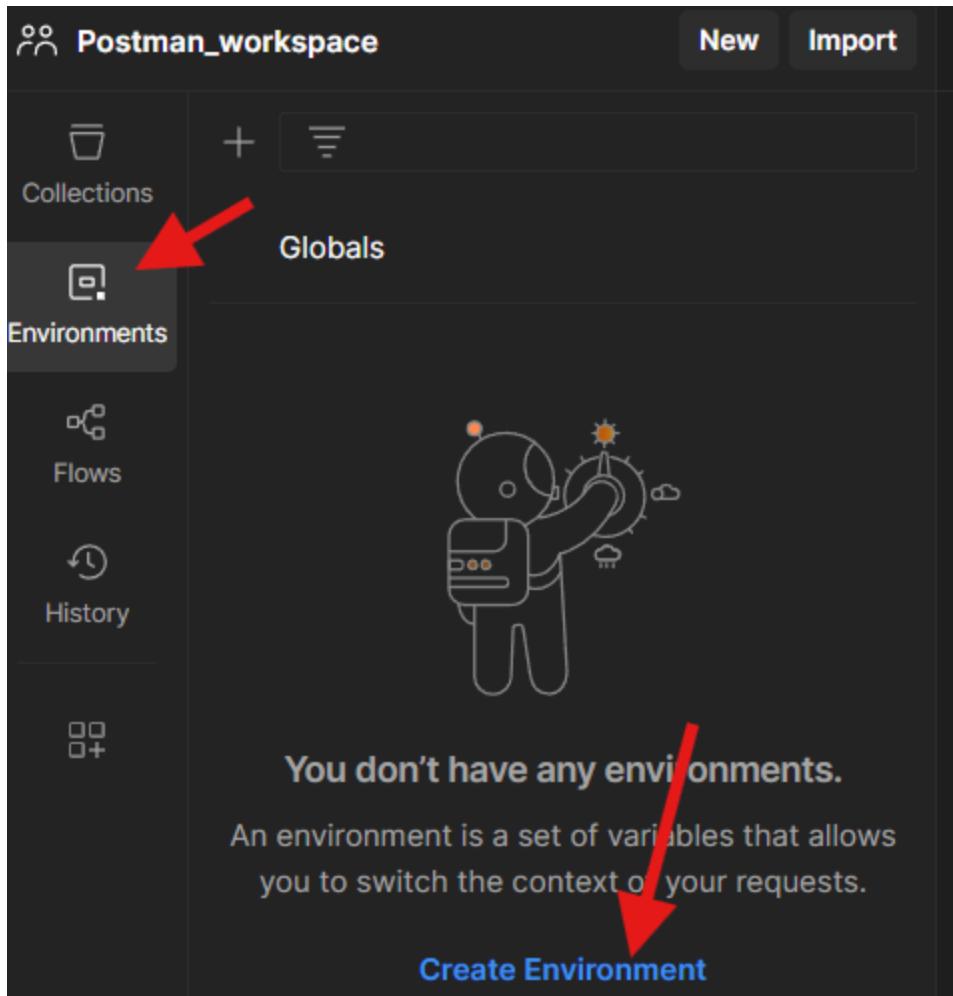
```
http  
CopyEdit  
POST /login
```

But this API exists in **different environments**:

Environment	URL
Dev	https://dev.api.myapp.com/login
QA/Test	https://test.api.myapp.com/login
Prod	https://api.myapp.com/login

Create Environment

We are copying the id and tokens to delete request. To avoid copying we can use environment.



The screenshot shows the Postman environment variables table and a POST request for 'createBooking'.

Postman_environment Variables:

Variable	Type	Initial value	Current value
First_name	default	MS	MS
Last_name	default	Dhoni	Dhoni

POST createBooking Request:

- URL: <https://restful-booker.herokuapp.com/booking>
- Method: POST
- Body (raw JSON):

```
{
  "firstname": "{{First_name}}",
  "lastname": "{{Last_name}}",
  "totalprice": 111,
  "depositpaid": true,
  "bookingdates": {
    "checkin": "2018-01-01",
    "checkout": "2019-01-01"
  },
  "additionalneeds": "Breakfast"
}
```

- Response status: 200 OK

To set an Environment variable choose the scripts, snippets.

The screenshot shows a DELETE request for 'deleteBooking' with a script in the Pre-request tab.

DELETE deleteBooking Request:

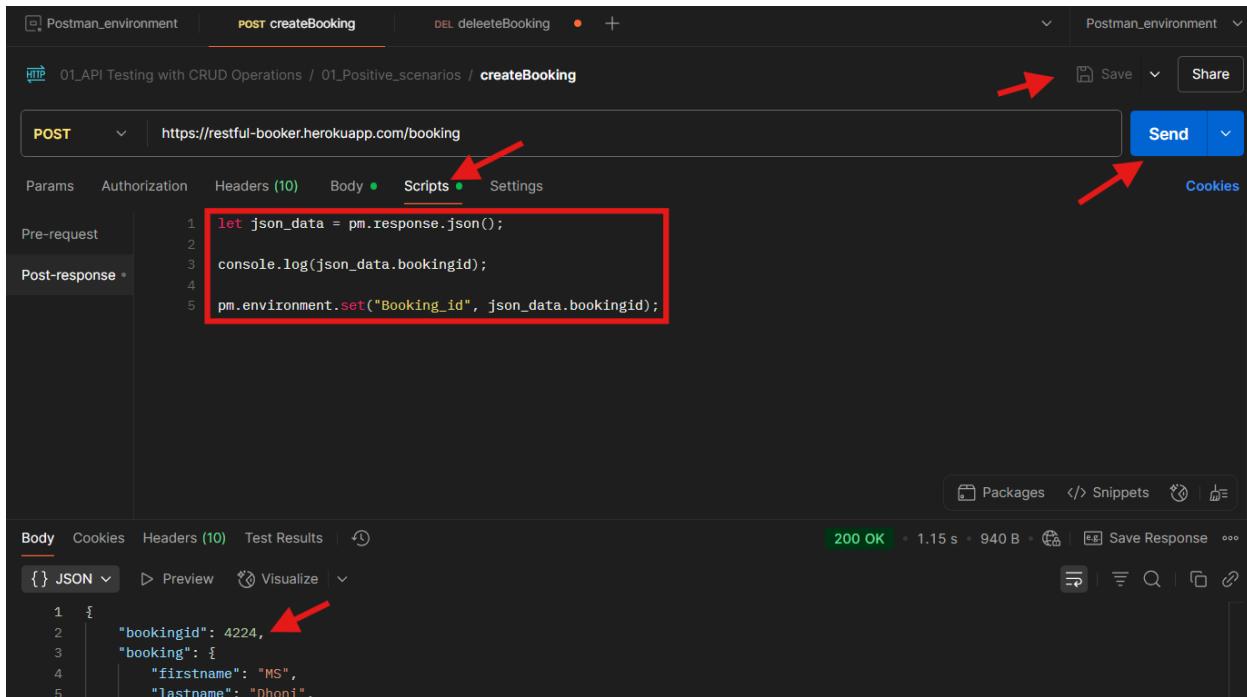
- URL: <https://restful-booker.herokuapp.com/booking/131>
- Method: DELETE
- Body (raw JSON):

```
pm.environment.set("variable_key", "variable_value");
```

A context menu is open over the script line, showing options for setting environment variables:

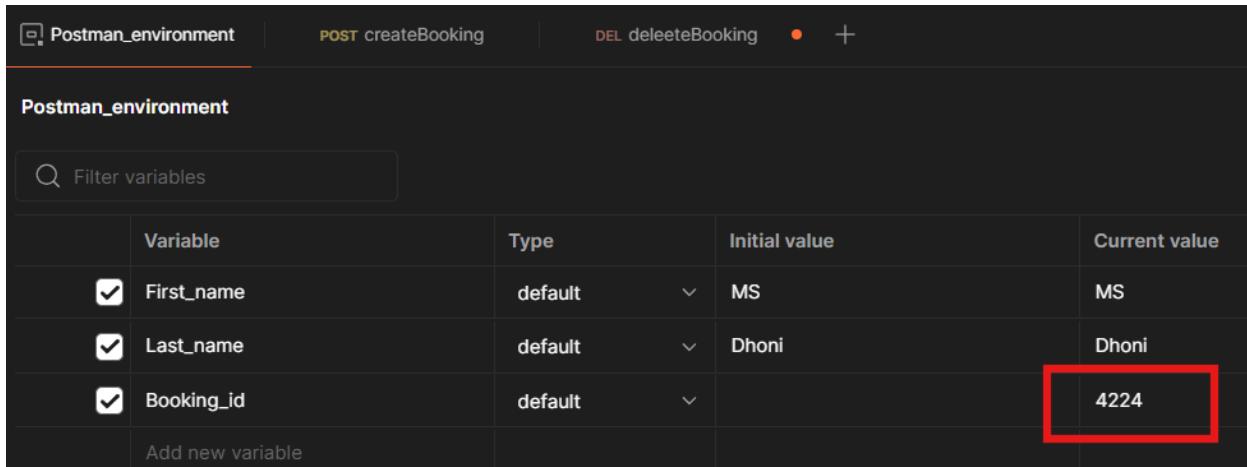
- Get a variable
- Set a global variable
- Set a collection variable
- Set an environment variable** (highlighted)
- Set a variable
- Clear a global variable
- Clear a collection variable
- Clear an environment variable
- Clear a local variable

In the Post Request that makes the response



```
let jsonData = pm.response.json();
console.log(jsonData.bookingId);
pm.environment.set("Booking_id", jsonData.bookingId);
```

Adds to the environment:



	Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/>	First_name	default	MS	MS
<input checked="" type="checkbox"/>	Last_name	default	Dhoni	Dhoni
<input checked="" type="checkbox"/>	Booking_id	default		4224
	Add new variable			

Every time id changes:

Postman environment

POST <https://restful-booker.herokuapp.com/booking>

Send

Params Authorization Headers (10) Body Scripts Settings

Pre-request

```
1 let json_data = pm.response.json();
2
3 console.log(json_data.bookingid);
4
5 pm.environment.set("Booking_id", json_data.bookingid);
```

Post-response

Body Cookies Headers (10) Test Results

200 OK 2.15 s 939 B Save Response

```
1 {
2   "bookingid": 701,
3   "booking": {
```

Postman_workspace

New Import

Postman_environment

	Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/>	First_name	default	MS	MS
<input checked="" type="checkbox"/>	Last_name	default	Dhoni	Dhoni
<input checked="" type="checkbox"/>	Booking_id	default		701

Now we can change the id of Get Booking by Id:

Postman_environment

GET https://restful-booker.herokuapp.com/booking/{{Booking_id}}

Send

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (10) Test Results

200 OK 662 ms 911 B Save Response

```
1 {
2   "firstname": "MS",
3   "lastname": "Dhoni",
4   "totalprice": 111,
```

To add the token to the environment variable:

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows 'Postman_workspace' with 'Collections', 'Environments', 'Flows', and 'History' sections.
- Request Details:**
 - Method: POST
 - URL: https://restful-booker.herokuapp.com/auth
 - Headers (11): Includes 'Content-Type: application/json'
 - Body: JSON
 - Scripts tab (highlighted with a red arrow): Contains the following code:

```
1 const jsonData = pm.response.json();
2
3 pm.environment.set("Token", jsonData.token);
```
 - Post-response tab: Contains a placeholder for response processing.
- Response Preview:** Shows a 200 OK status with a response body containing a token: "token": "83fc757088b2670".
- Bottom Right:** Buttons for 'Save', 'Share', 'Send', and 'Snippets'.

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows 'Postman_workspace' with 'Collections', 'Environments' (highlighted with a red arrow), 'Flows', and 'History' sections.
- Request Details:** Same as the previous screenshot, showing the tokenGenerator POST request.
- Environment Variables:**
 - Environment: Postman_environment (highlighted with a red arrow).
 - Variables table:

Variable	Type	Initial value	Current value
First_name	default	MS	MS
Last_name	default	Dhoni	Dhoni
Booking_id	default		701
Token	default		83fc757088b2670

Token actually keeps on changing. Fixed response for idempotent.

Now You don't have to copy and paste the token and id. Example given below:

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows '01_API Testing with CRUD Operations' with '01_Positive_scenarios' and '02_Negative_scenarios' sections. The 'POST createBooking' request is highlighted with a red arrow.
- Request Details:**
 - Method: POST
 - URL: https://restful-booker.herokuapp.com/booking
 - Headers (10): Includes 'Content-Type: application/json'
 - Body: JSON
 - Scripts tab (highlighted with a red arrow): Contains the following code:

```
1 let jsonData = pm.response.json();
2
3 console.log(jsonData.bookingId);
4
5 pm.environment.set("Booking_id", jsonData.bookingId);
```
 - Post-response tab: Contains a placeholder for response processing.
- Response Preview:** Shows a 200 OK status with a response body containing a booking ID: "bookingId": 632.
- Bottom Right:** Buttons for 'Save', 'Share', 'Send' (highlighted with a red arrow), and 'Cookies'.

The image displays three screenshots of the Postman application interface, showing API requests and their responses.

Screenshot 1: GET Request to https://restful-booker.herokuapp.com/booking/{{Booking_id}}

- Request Details:** Method: GET, URL: https://restful-booker.herokuapp.com/booking/{{Booking_id}}. The "Send" button is highlighted with a red arrow.
- Headers:** Params, Authorization, Headers (7), Body, Scripts, Settings.
- Query Params:** Key, Value, Description, Bulk Edit.
- Body:** JSON, Preview, Visualize. Response status: 200 OK, 266 ms, 911 B. Response body (JSON):


```

1 {
2   "firstname": "MS",
3   "lastname": "Dhoni",
4   "totalprice": 111,
      
```

Screenshot 2: POST Request to https://restful-booker.herokuapp.com/auth

- Request Details:** Method: POST, URL: https://restful-booker.herokuapp.com/auth. The "Send" button is highlighted with a red arrow.
- Headers:** Params, Authorization, Headers (11), Body, Scripts, Settings.
- Pre-request:** const json_data = pm.response.json();
- Post-response:** pm.environment.set("Token", json_data.token);
- Body:** JSON, Preview, Visualize. Response status: 200 OK, 268 ms, 770 B. Response body (JSON):


```

1 {
2   "token": "a4a0685ef7a4f3a"
3 }
      
```

Screenshot 3: PUT Request to https://restful-booker.herokuapp.com/booking/{{Booking_id}}

- Request Details:** Method: PUT, URL: https://restful-booker.herokuapp.com/booking/{{Booking_id}}. The "Send" button is highlighted with a red arrow.
- Headers:** Params, Authorization, Headers (12), Body, Scripts, Settings.
- Headers (9 hidden):**
 - Content-Type: application/json
 - Accept: application/json
 - Cookie: token={{Token}}
- Body:** JSON, Preview, Visualize. Response status: 200 OK, 337 ms, 910 B. Response body (JSON):


```

1 {
2   "firstname": "James",
3   "lastname": "Brown",
4   "totalprice": 111,
5   "depositpaid": true,
      
```

Its showing james brown because

The screenshot shows the Postman interface with a successful API call. The URL is `https://restful-booker.herokuapp.com/booking/{{Booking_Id}}`. The response status is 200 OK, with a response time of 337 ms and a response size of 910 B. The raw JSON response body is identical to the request body:

```

1 {
2   "firstname": "James",
3   "lastname": "Brown",
4   "totalprice": 111,
5   "depositpaid": true,
6   "bookingdates": {
7     "checkin": "2018-01-01",
8     "checkout": "2019-01-01"
9   },
10  "additionalneeds": "Breakfast"
11 }

```

After Updating the firstname and lastname

The screenshot shows the Postman interface after updating the booking. A red arrow points to the "Send" button, indicating the action taken to trigger the update. Another red arrow points to the "lastname" field in the request body, which has been modified to "Dhoni". The response status is 200 OK, with a response time of 1.99 s and a response size of 907 B. The raw JSON response body shows the updated values:

```

1 {
2   "firstname": "MS",
3   "lastname": "Dhoni",
4   "totalprice": 111,
5   "depositpaid": true,
6   "bookingdates": {
7     "checkin": "2018-01-01",
8     "checkout": "2019-01-01"
9   },
10  "additionalneeds": "Breakfast"
11 }

```

The screenshot shows the Postman interface with a DELETE request to `https://restful-booker.herokuapp.com/booking/{{Booking_id}}`. The 'Send' button is highlighted with a red arrow. In the Headers table, two entries are shown: `Content-Type: application/json` and `Cookie: token={{Token}}`. The response tab shows a successful `201 Created` status.

If you're using a tool like Postman, you can create environments and just change the **base URL variable**, like this:

```
js
CopyEdit
{{base_url}}/login
```

Then you define `base_url` for each environment, and switch easily.

⌚ Why this is helpful:

- You don't have to rewrite your tests every time.
- You can safely test without affecting real users.
- You can switch environments quickly and test in different situations.

Console

What is the **Console** in Postman?

The **console** in Postman is like a small window where you can **see what's really happening** when you send an API request.

✓ Why do we use it?

1. See details of your request

- What you sent
- What you got back
- Any errors

2. Check if variables are working

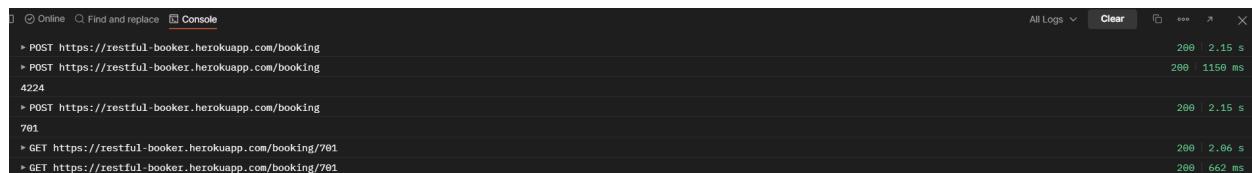
- You can print variables like tokens or URLs to make sure they're correct.

3. Find problems

- ↳ ⚡ • If something is not working, the console shows what went wrong (like wrong URL, missing data, etc.)

4. Debug your test scripts

- You can write `console.log("Hello")` in your script and see it in the console — just like checking if something is working.



A screenshot of the Postman interface focusing on the 'Console' tab. The tab bar at the top shows 'Online' and 'Find and replace'. The 'Console' tab is active, indicated by a blue border. Below the tabs, there is a list of API requests and their responses. The requests include POST and GET methods to 'https://restful-booker.herokuapp.com/booking'. The responses show status codes (200), execution times (e.g., 2.15 s, 1150 ms), and the raw JSON data returned from the server. The JSON data includes fields like 'id', 'first_name', 'last_name', 'total_price', 'check_in', 'check_out', 'adults_count', and 'children_count'. The interface has a clean, modern design with a light gray background and white text.

CTRL + ALT + C to open the console.

Simple Example:

Let's say your test script has this line:

```
js
CopyEdit
console.log("Token is:", pm.environment.get("token"));
```

When you run the request, the console will show:

```
csharp
CopyEdit
Token is: abc123xyz
```

1. Which of the following HTTP methods is *not* idempotent?

- A. GET
- B. PUT
- C. DELETE
- D. POST

 **Correct Answer:** D. POST

 **Explanation:** POST typically creates a new resource each time it's called, which changes the state, so it's not idempotent.

2. What does it mean if an HTTP method is *idempotent*?

- A. It never returns an error.
- B. It always returns the same response format.
- C. It has the same effect no matter how many times it's called.
- D. It only works on DELETE operations.

 **Correct Answer:** C. It has the same effect no matter how many times it's called.

 **Explanation:** Idempotent methods can be safely repeated without changing the server's state after the initial call.

3. Which HTTP method would you use to partially update a user's profile?

- A. GET
- B. PUT
- C. POST
- D. PATCH

 **Correct Answer:** D. PATCH

 **Explanation:** PATCH is used to update part of a resource — like just the user's email or bio.

4. Which environment is typically used as a "final test" before going live in API testing?

A. Development

 Explain  Ask AI  Comment  Text **B** **I** **U** **S** **</>** \sqrt{x}   A 

C. Staging

D. QA/Test

 **Correct Answer:** C. Staging

 **Explanation:** Staging is like a dress rehearsal — used for final testing in a near-live setup.

Assertions in API Testing

What is an Assertion?

An assertion is a statement that validates whether a specific condition is true or false during testing.

In API testing, assertions help confirm that the response from an API meets expected criteria.

Why Are Assertions Important?

- Verify correctness of API responses
- Catch bugs and issues early
- Validate data integrity and response format
- Ensure performance and reliability

🔍 Common Types of Assertions in API Testing

Assertion Type	What It Checks	Example
>Status Code	Is the HTTP status code correct?	200 OK, 404 Not Found
📝 Response Body	Does the body contain expected data or text?	Includes "success" or user_id
🌐 Headers	Are required headers present and correct?	Content-Type, Authorization
⌚ Response Time	Is the API responding within acceptable time limits?	< 1000ms
🔒 Authentication	Is access restricted/protected as expected?	401 Unauthorized

🛠️ How Assertions Work in Postman

In Postman, you write assertions using the `pm` object (Postman's scripting interface).

Example:

```
javascript
CopyEdit
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

This checks that the response has a 200 status code.

✳️ Assertion Libraries in Use

Postman uses ChaiJS (a JavaScript assertion library), which allows:

- BDD-style assertions like `to.have`, `to.be.below`, `to.include`, etc.

💡 Best Practices for Using Assertions

- Use **clear test names** (describe what you're testing)
- Check **status code, response body, headers, and performance**
- Cover **positive and negative scenarios**

Postman Test Script Notes

✓ 1. Status Code Validation

```
javascript
CopyEdit
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

- **Purpose:** Verifies the response has a 200 OK status.
- **Use Case:** Ensures the API call was successful.

🔍 2. Response Body Content Check

```
javascript
CopyEdit
pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("Dhoni");
});
```

- **Purpose:** Checks if the **response body** contains the text **"Dhoni"**.
- **Use Case:** Confirms specific **data/content** is returned by the API.

📝 3. Header Validation

```
javascript
CopyEdit
pm.test("Content-Type is present", function () {
    pm.response.to.have.header("Content-Type", "application/json; charset=utf-8");
});
```

- **Purpose:** Verifies the **Content-Type header** is correct.
- **Use Case:** Ensures the response is in **JSON format**.

The screenshot shows the Postman workspace interface. On the left, there's a sidebar with 'Collections', 'environments', 'Flows', and 'History'. The main area displays a 'POST createBooking' request under '01_Positive_scenarios'. The 'Tests' tab in the sidebar is open, showing several assertions related to the response status code and body. A red arrow points from the bottom right of the 'Tests' sidebar towards the 'Status code: Code is 200' assertion.

```
let json_data = pm.response.json();
// console.log(json_data.bookingId);
pm.environment.set("Booking_id", json_data.bookingId);

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Top Panel:

Middle Panel:

Bottom Panel:

The screenshots demonstrate the execution of API tests in Postman. The top panel shows a successful POST operation for creating a booking. The middle panel shows a successful GET operation for retrieving a booking by ID. The bottom panel shows a failed GET operation for retrieving a booking by an invalid ID, resulting in a 404 Not Found error.

The screenshot shows the Postman application interface. At the top, there are several tabs: 'Postman_en', 'POST createBool', 'DEL deleteBool', 'GET getBooking1', 'POST tokenGene', 'PUT updateBool', 'GET getBooking1', 'GET getBooking1'. Below the tabs, the URL 'https://restful-booker.herokuapp.com/booking/167' is entered into the address bar. On the left, there are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Scripts', and 'Settings'. The 'Scripts' tab is currently selected. Under 'Pre-request', there is a script block with three lines of code: 'pm.test("Status code is 200", function () {', '2 | pm.response.to.have.status(200);', '3});'. The 'Test Results' tab is selected, showing '0/1' results. A red 'FAILED' button is visible. Below it, the message 'Status code is 200 | Assertion Error: expected response to have status code 200 but got 404' is displayed. The bottom right corner shows a 'Save Response' button.

1. What is the primary purpose of an assertion in API testing?

- A. To create API endpoints
- B. To encrypt the API request
- C. To validate that a condition is true or false
- D. To generate fake API responses

| Correct Answer: C — Assertions validate whether a condition is true or false.

2. Which of the following assertions would be used to check API response speed?

- A. Response Body
- B. Status Code
- C. Response Time
- D. Headers

| Correct Answer: C — Response Time assertions ensure the API responds within acceptable limits.

3. In Postman, which JavaScript object is commonly used to write test assertions?

- A. `assert`
- B. `pm`
- C. `chai`
- D. `http`

| Correct Answer: B — Postman uses the `pm` object to write assertions.

4. What does the following Postman test script validate?

```
javascript
CopyEdit
pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("Dhoni");
});
```

- A. The response is in JSON format
- B. The response status is 200
- C. The response body contains the text "Dhoni"
- D. The API took less than 1000ms to respond

Correct Answer: C — This assertion checks for the presence of "Dhoni" in the response body.